Architecting Secure Serverless and Containerized Applications

Josh Kahn
AWS
Tech Leader, Serverless

Jimmy Ray
AWS
Sr. Developer Advocate, AWS Kubernetes

“Security is everyone’s job”

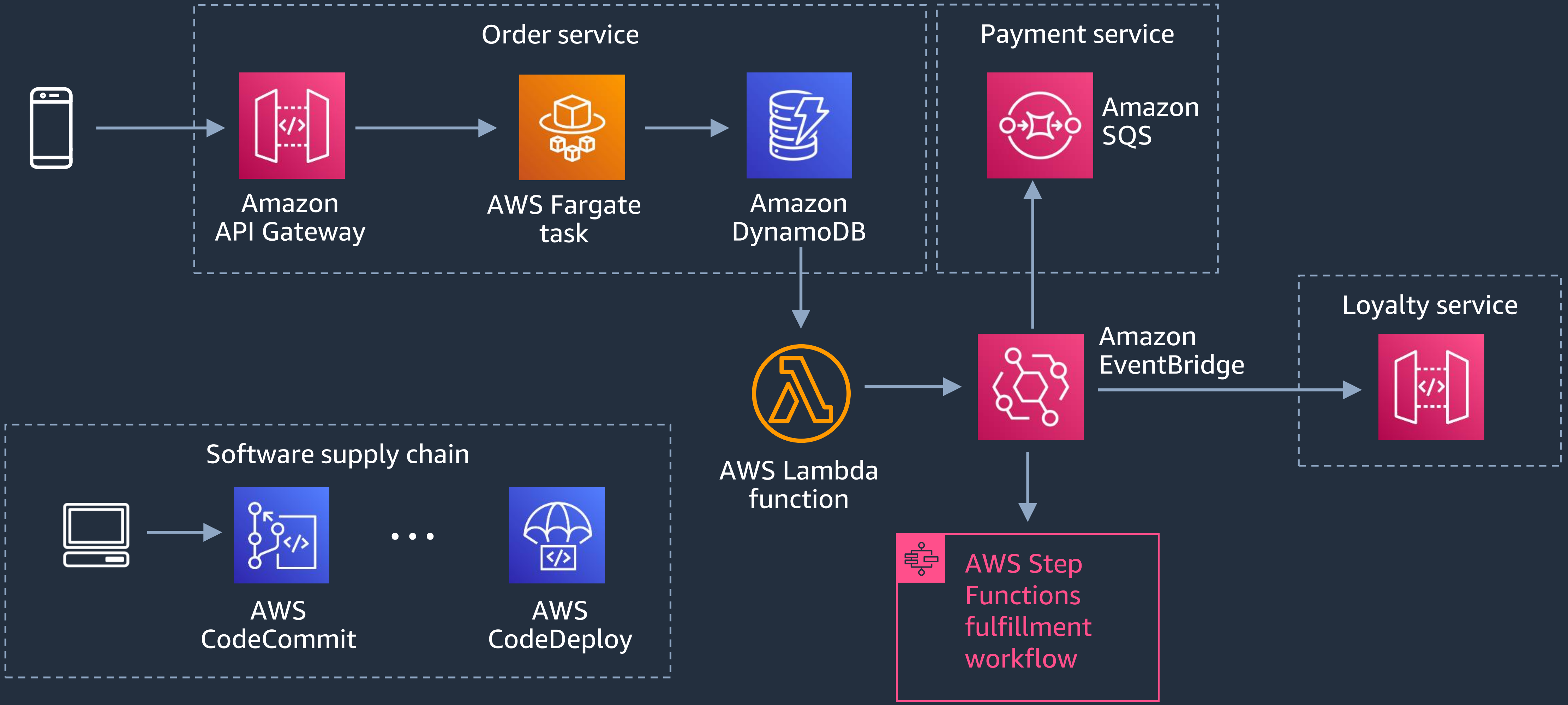
Werner Vogel

AWS CTO

Four principles to securing modern applications

1. Understand **shared responsibility model**
2. Grant **least privilege**
3. Implement **defense in depth**
4. Secure your **software supply chain**

Building modern applications with microservices



Security considerations for microservices

- More transient and dynamic

- More distributed and complex
  - More services interdependencies over network
  - Scheduling / scaling / resource management

- Isolation is similar to virtual machines, but different:
  - May share a kernel
  - May share a network and a network interface

(Subset of) Options to build microservices

AWS Lambda

AWS Fargate

Amazon ECS

Amazon EKS

Comparison of operational responsibility

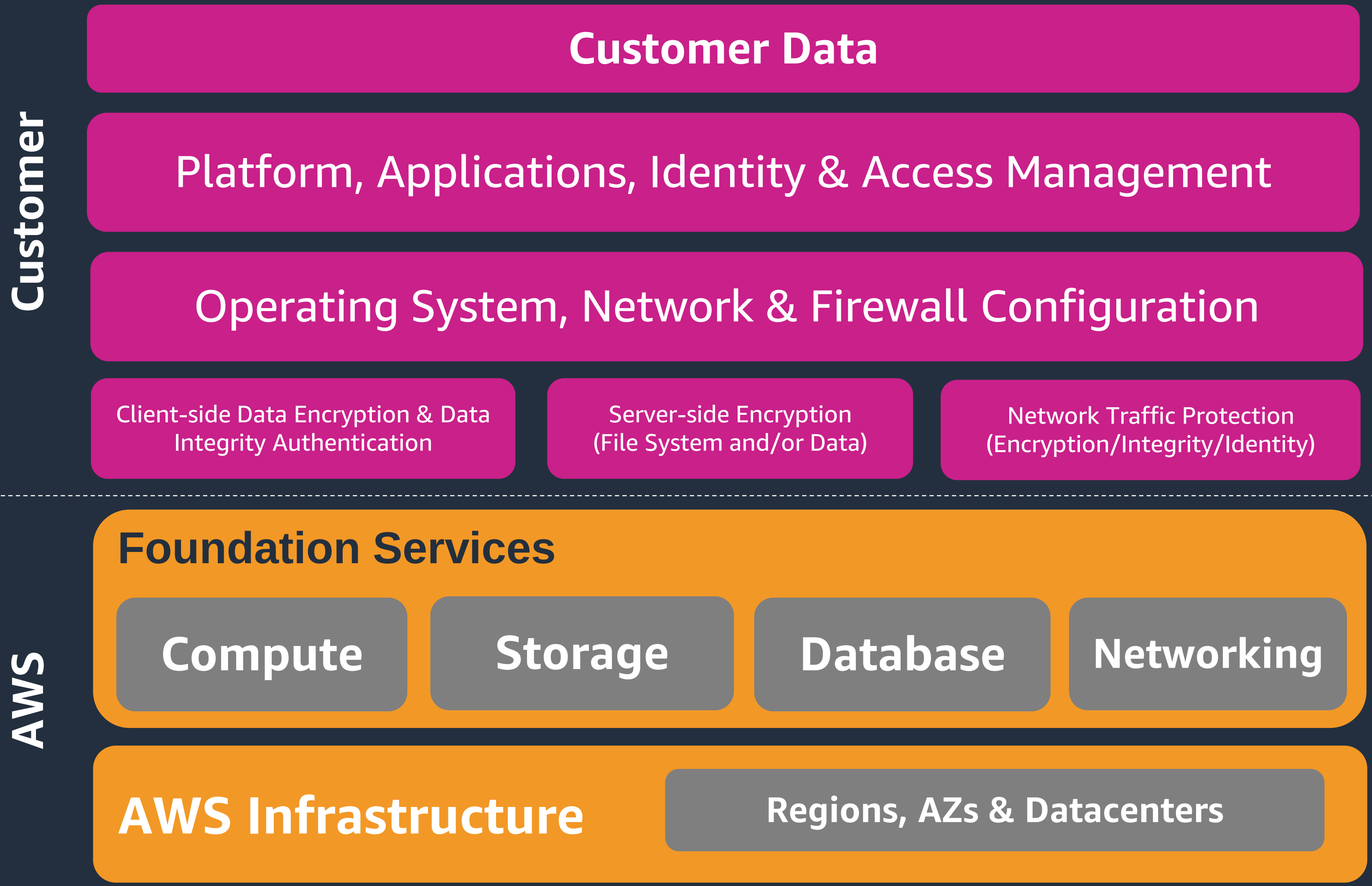
More opinionated

	AWS manages	Customer manages
Lambda Serverless functions	• Data source integrations • Physical hardware, software, networking, and facilities • Provisioning	• Application code
Fargate Serverless containers	• Container orchestration, provisioning • Cluster scaling • Physical hardware, host OS/kernel, networking, and facilities	• Application code • Data source integrations • Security config and updates, network config, management tasks
Amazon ECS/ Amazon EKS Container management as a service	• Container orchestration control plane • Physical hardware software, networking, and facilities	• Application code • Data source integrations • Work clusters • Security config and updates, network config, firewall, management tasks
Amazon EC2 Infrastructure as a service	• Physical hardware software, networking, and facilities	• Application code • Data source integrations • Scaling • Security config and updates, network config, management tasks • Provisioning, managing scaling and patching of servers

Less opinionated

Security Principle #1: Shared Responsibility with AWS

Security principle #1: Shared Responsibility with AWS



Customer

Customer Data

Platform, Applications, Identity & Access Management

Operating System, Network & Firewall Configuration

Client-side Data Encryption & Data Integrity Authentication

Server-side Encryption (File System and/or Data)

Network Traffic Protection (Encryption/Integrity/Identity)

AWS

Foundation Services

Compute

Storage

Database

Networking

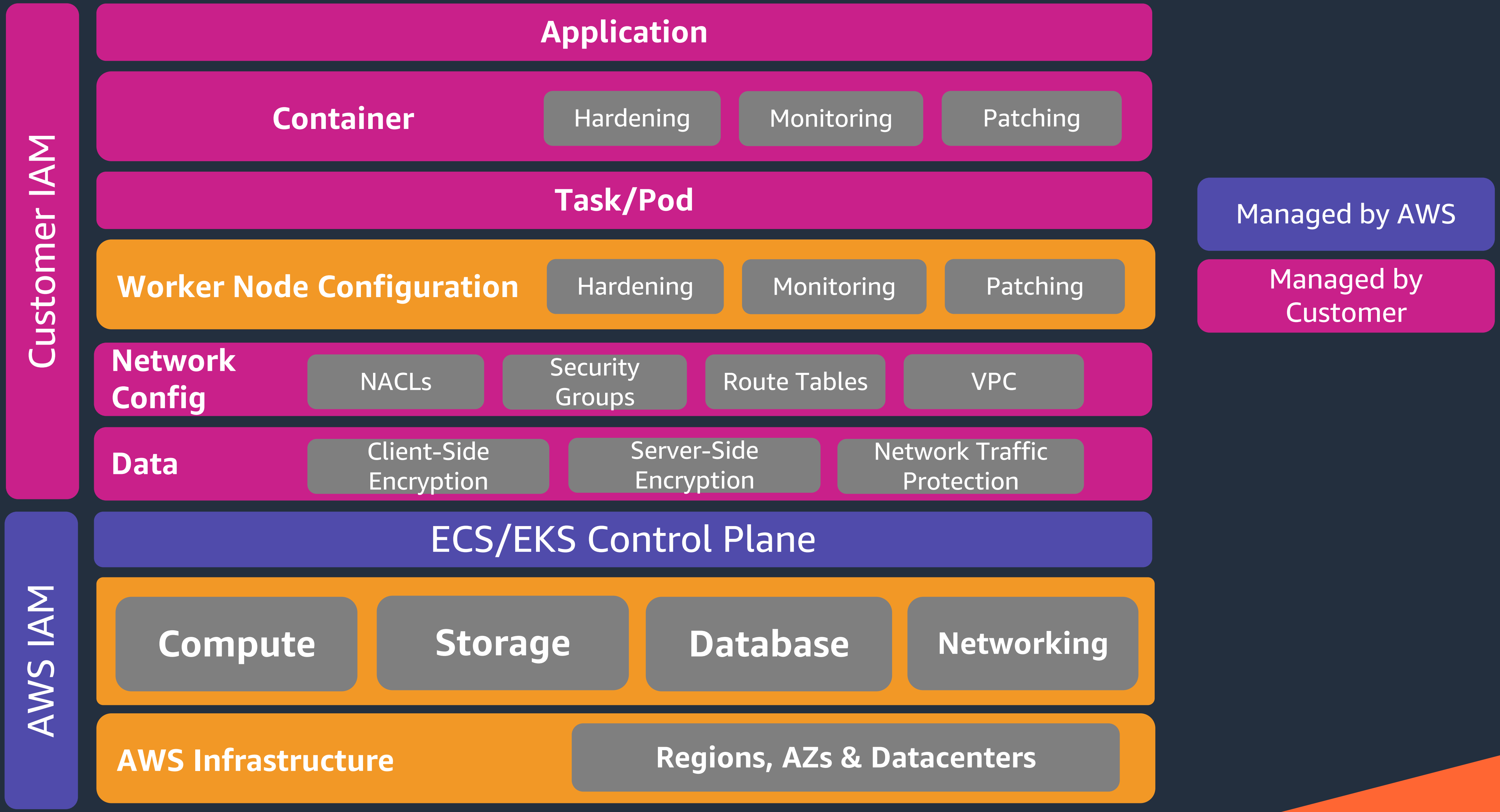
AWS Infrastructure

Regions, AZs & Datacenters

Customers are responsible for their security and compliance **IN** the Cloud

AWS is responsible for the security **OF** the Cloud

Responsibilities change with AWS Fargate



Customer IAM

Application

Container: Hardening, Monitoring, Patching

Task/Pod

Worker Node Configuration: Hardening, Monitoring, Patching

Network Config: NACLs, Security Groups, Route Tables, VPC

Data: Client-Side Encryption, Server-Side Encryption, Network Traffic Protection

AWS IAM

ECS/EKS Control Plane

Compute, Storage, Database, Networking

AWS Infrastructure: Regions, AZs & Datacenters

Managed by AWS

Managed by Customer

Security benefits of AWS Fargate

We do more, you do less.

- Patching (OS, Docker, Amazon ECS agent, etc.)
- Task isolation
- No --privileged mode for containers
- AES-256 Server side encryption of ephemeral storage

Container orchestration

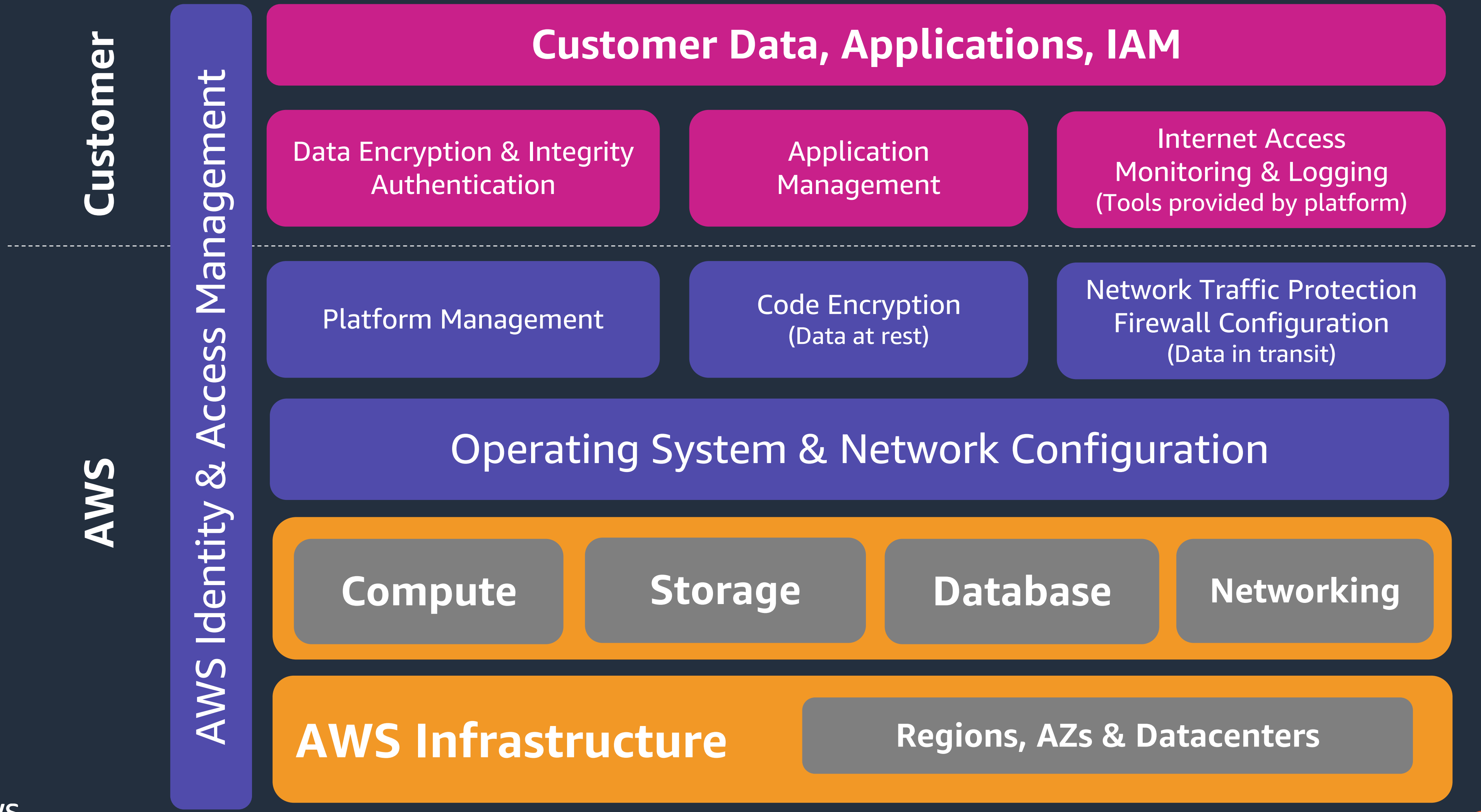
AWS managed control planes

- Elastic Container Service (ECS)
- Elastic Kubernetes Services (EKS)
- Responsible for managing the scheduling and lifecycle of containers

Data plane

- Self managed EC2
- Managed node groups (EKS only)
- Fargate (ECS and EKS)

With Serverless, AWS takes an even greater share of responsibility



Customer

AWS

AWS Identity & Access Management

Customer Data, Applications, IAM

Data Encryption & Integrity Authentication

Application Management

Internet Access
Monitoring & Logging
(Tools provided by platform)

Platform Management

Code Encryption
(Data at rest)

Network Traffic Protection
Firewall Configuration
(Data in transit)

Operating System & Network Configuration

Compute

Storage

Database

Networking

AWS Infrastructure

Regions, AZs & Datacenters

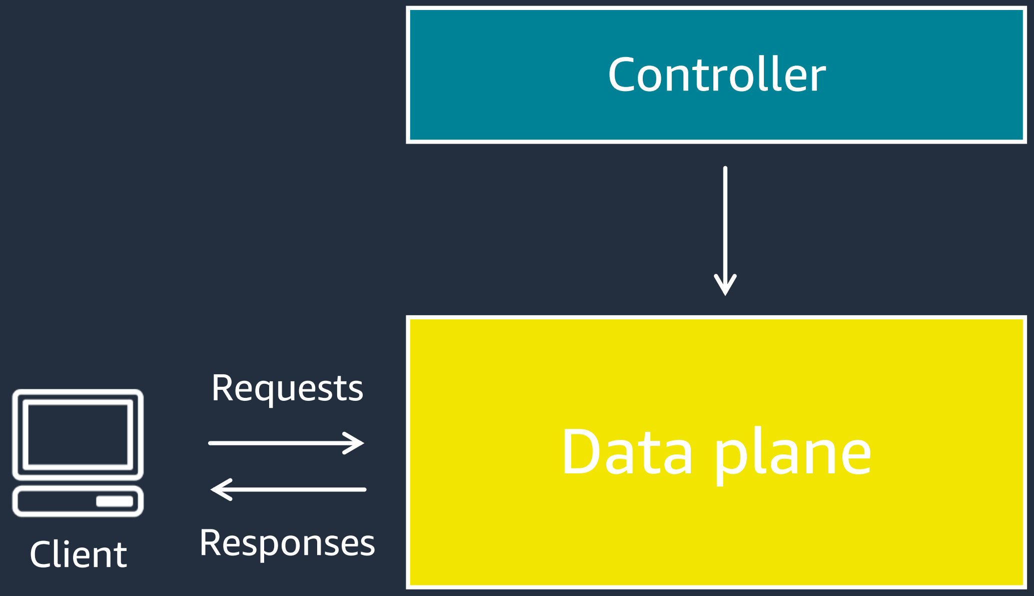
Lambda service composed of control plane and data plane

Control Plane

- Management APIs, such as:
  - `CreateFunction`
  - `UpdateFunctionCode`
- Requires IAM permission to access

Data Plane

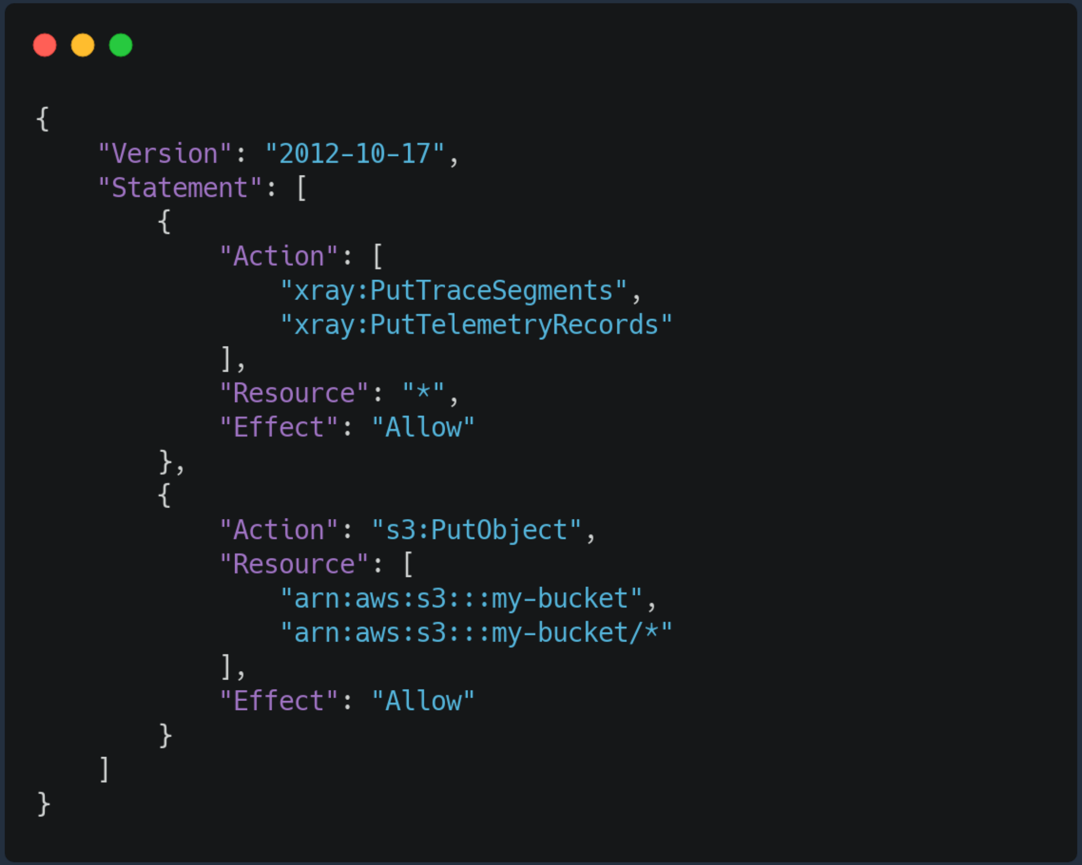
- Invoke Lambda function via `Invoke`
- Requires IAM permission or resource policy
- When invoked, data plane runs code on:
  - Existing execution environment, if exists
  - New environment, after allocation



Security Principle #2: Least Privilege

Security principle #2: Least Privilege

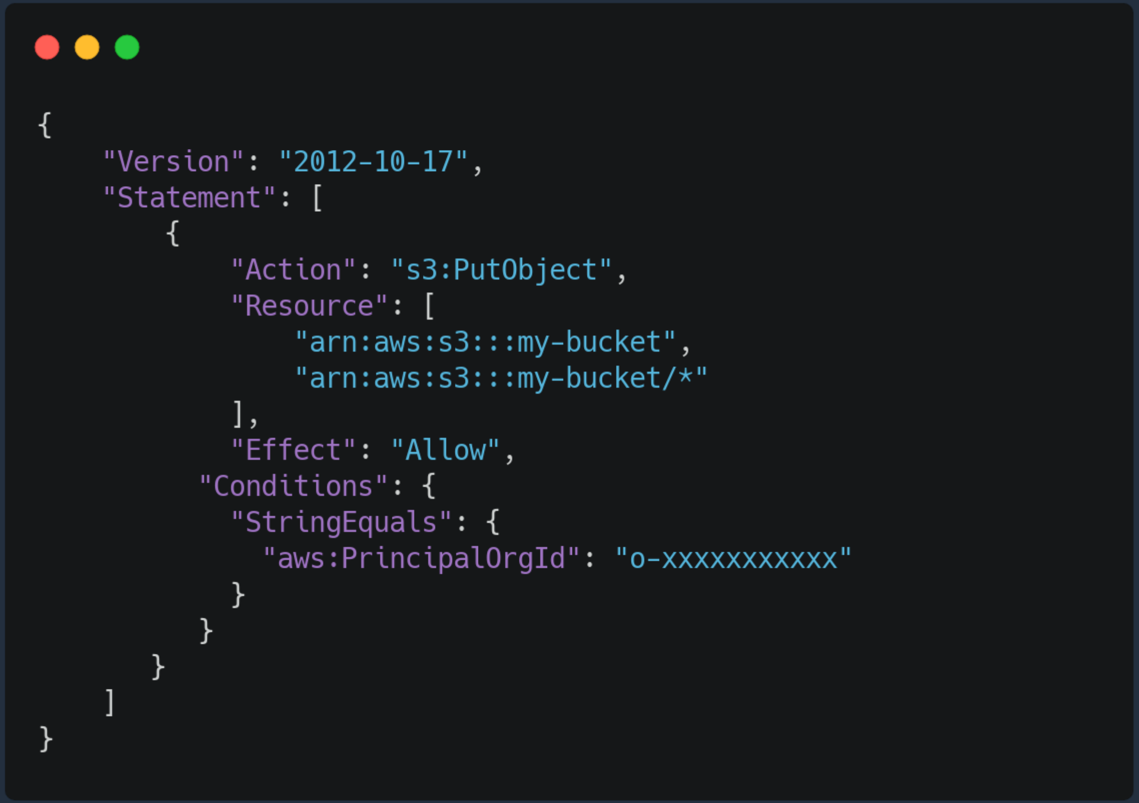
- Grant only the essential privileges needed to perform intended work
- Attach to compute via execution role
  - Prefer unique role per function or task
  - Enforce permission boundaries
- Be specific: identify limited set of resources and actions allowed
  - Scrutinize use of “*”



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::my-bucket",
                "arn:aws:s3:::my-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Use AWS IAM to assign and audit fine-grained permissions

- IAM roles can be assigned to:
    - ECS Tasks
    - Kubernetes Pods
    - Lambda Functions
    - Step Functions Workflows
    - … and more …
- Allow (or deny) access to AWS APIs (management, data planes)
- Periodically audit access
    - AWS Access Advisor
    - Amazon CloudTrail Insights
    - Kubernetes audit log/CloudWatch



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::my-bucket",
                "arn:aws:s3:::my-bucket/*"
            ],
            "Effect": "Allow",
          "Conditions": {
            "StringEquals": {
              "aws:PrincipalOrgId": "o-xxxxxxxxxxx"
            }
          }
        }
    ]
}
```

User access

Apply principles of least privilege.

- Authenticate all user access to hosts and containers.
- Implement IAM policies and roles to restrict access to only required services.
- Restrict access and write permissions to image registry.

AWS Identity and Access Management (IAM)

Permissions

Role

AWS CloudTrail

Security, Governance, and Oversight = Authentication + Authorization + Audit/Log

Security Principle #3: Practice Defense in Depth

Common vectors of attack

App Vulnerabilities

SQL Injection

Cross-site Scripting (XSS)

OWASP Top 10

Common Vulnerabilities and Exposures (CVE)

Dependencies

Libraries

Distributions

Base Images

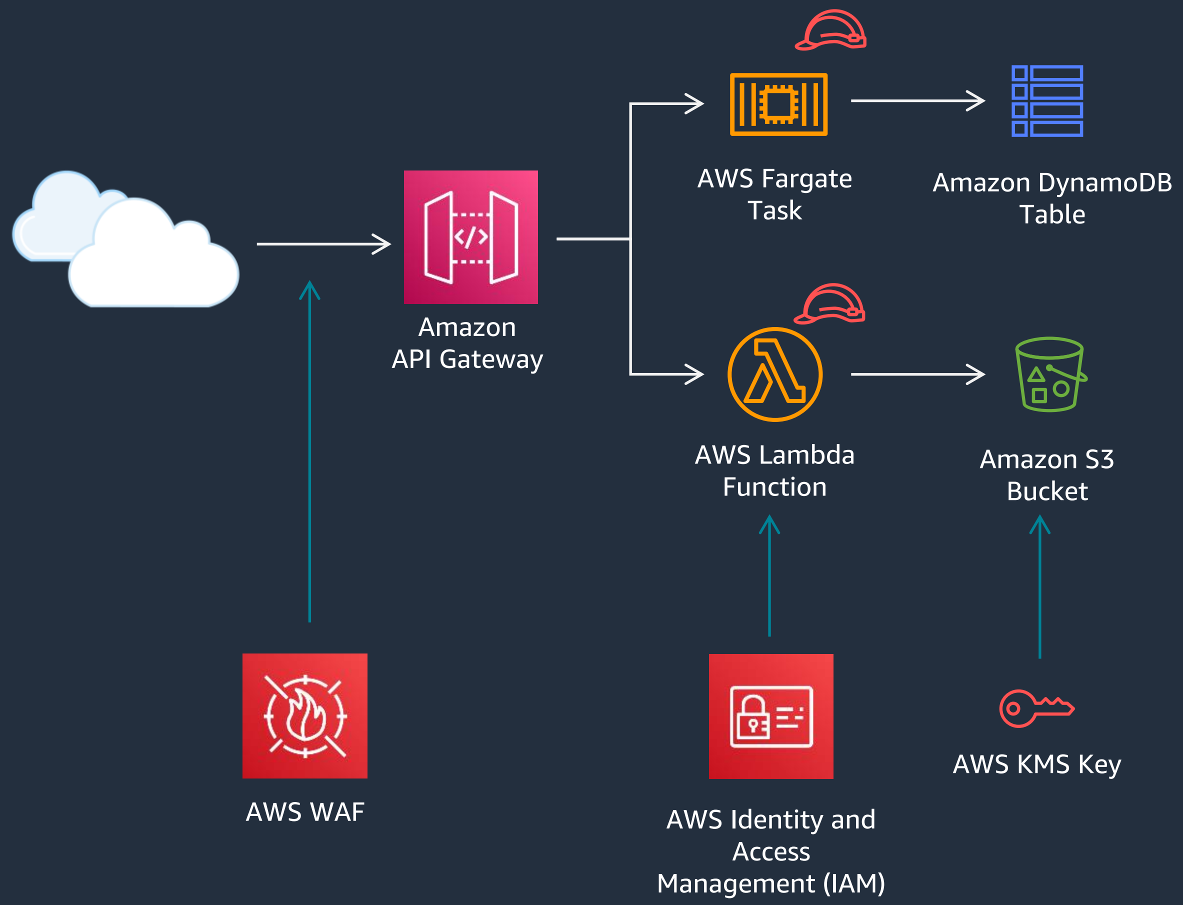
Host / Network

Patching

Network Segmentation

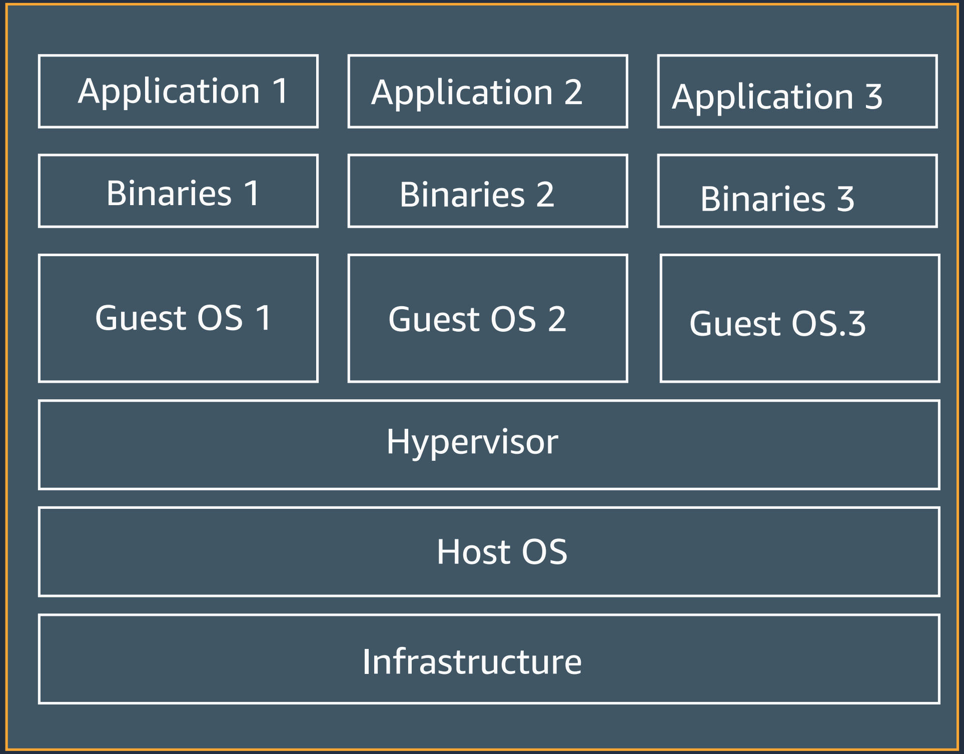
Security principle #3: Practice Defense in Depth

- Implement multiple, redundant measures across system to address common attack vectors
- Leverage AWS managed services and integrations
- Consider service features, e.g. backup and encryption



Container versus Virtual Machine

Virtual Machine



Application 1 | Application 2 | Application 3

Binaries 1 | Binaries 2 | Binaries 3

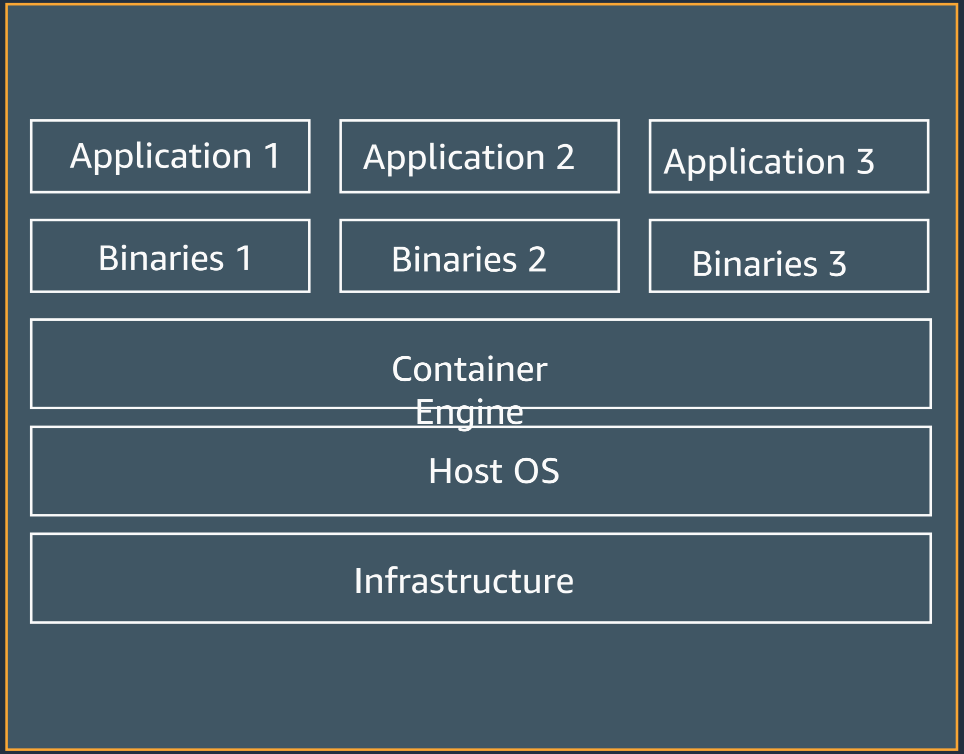
Guest OS 1 | Guest OS 2 | Guest OS.3

Hypervisor

Host OS

Infrastructure

Containers



Application 1 | Application 2 | Application 3

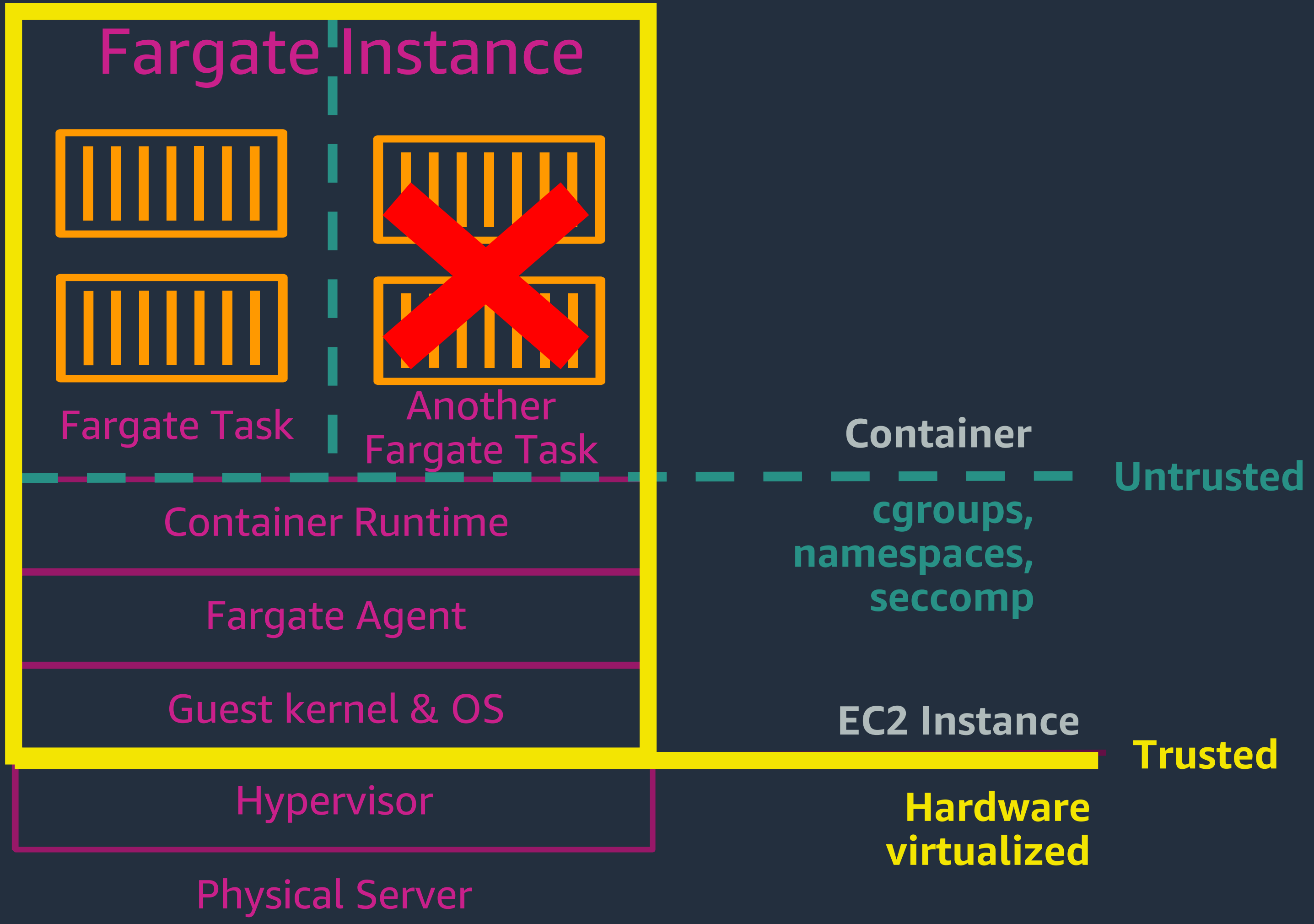
Binaries 1 | Binaries 2 | Binaries 3

Container Engine

Host OS

Infrastructure

One & only one task per EC2 instance



Containers: Runtime security options

- Containers run as processes on the Linux kernel
- Linux options:
  - cgroups
  - namespaces
  - Linux capabilities
  - seccomp*
  - AppArmor*
  - SELinux*

- 3rd party and open source security options include:
  - Aqua
  - Falco (CNCF project)
  - PA Primsa
  - Redhat StackRox
  - Sysdig Secure

* Not applicable to serverless containers (Fargate)

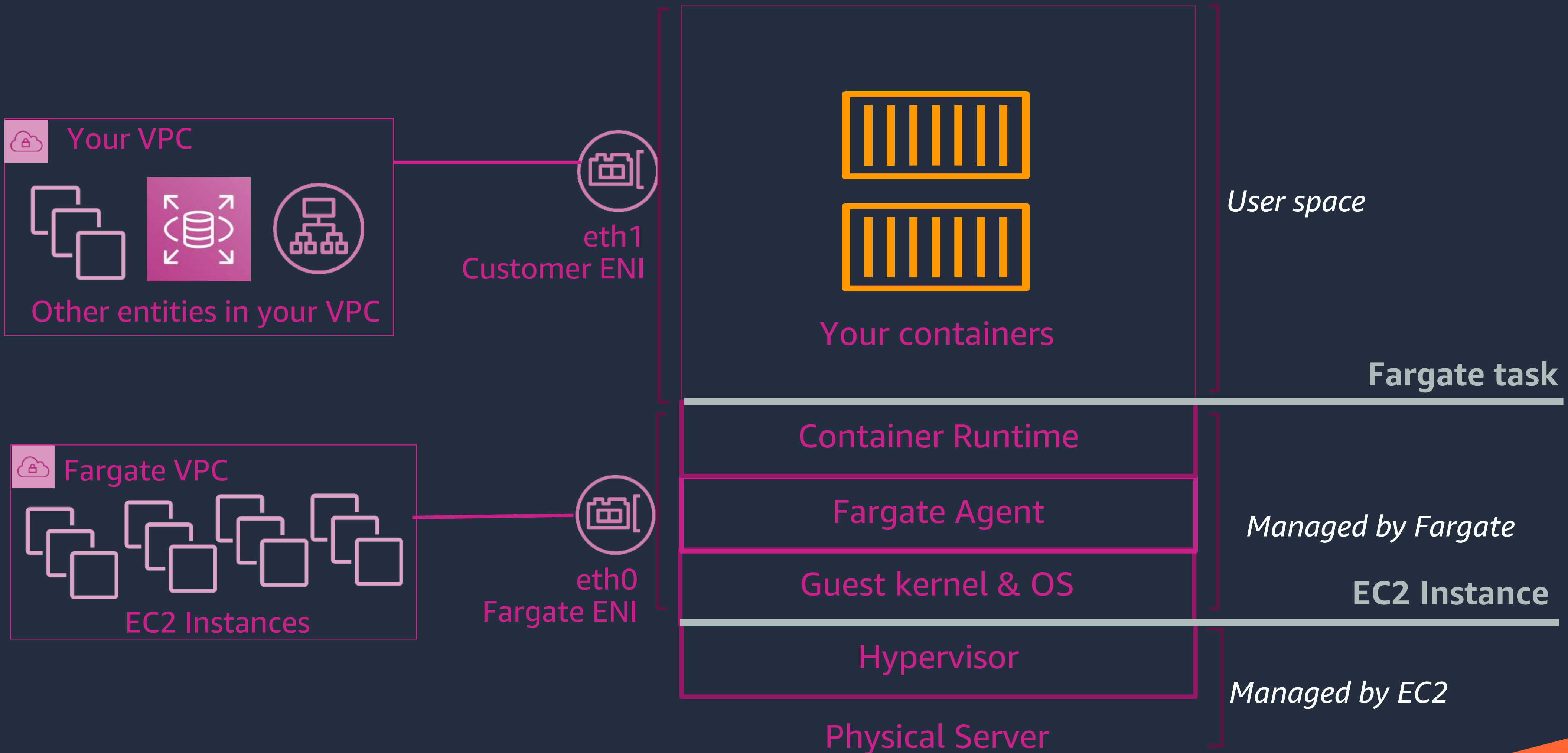
Containers: Network security options

- Restrict communication between:
  - Pods and Tasks
  - Containerized applications and other resources that run within or outside the VPC
- Encrypt traffic between:
  - Pods, Tasks, Instances, Lambda functions (future)
  - AWS load balancers and tasks/pods

Service specific options

- EKS
  - Kubernetes Network Policies
  - Security Groups for Pods
  - App Mesh (TLS & mTLS)
  - SSL/TLS (load balancing/ingress)
- ECS
  - Security Groups for Tasks
  - App Mesh (TLS & mTLS)
  - SSL/TLS (load balancing)

Fargate networking: A deeper look



Fargate: Process isolation

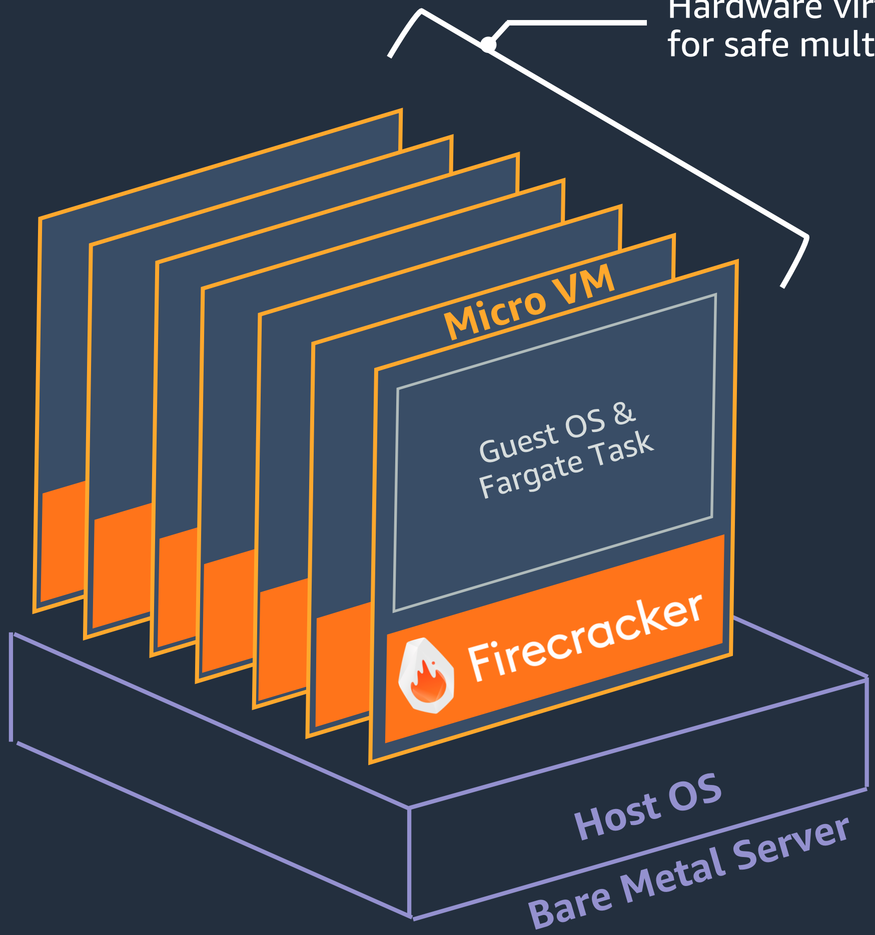
Fargate implements a shared nothing architecture

- Disk
- Memory
- CPU
- Network

Each task/pod runs as a separate virtual machine (EC2 or Firecracker)

Both VM types provide a hard security boundary

Firecracker

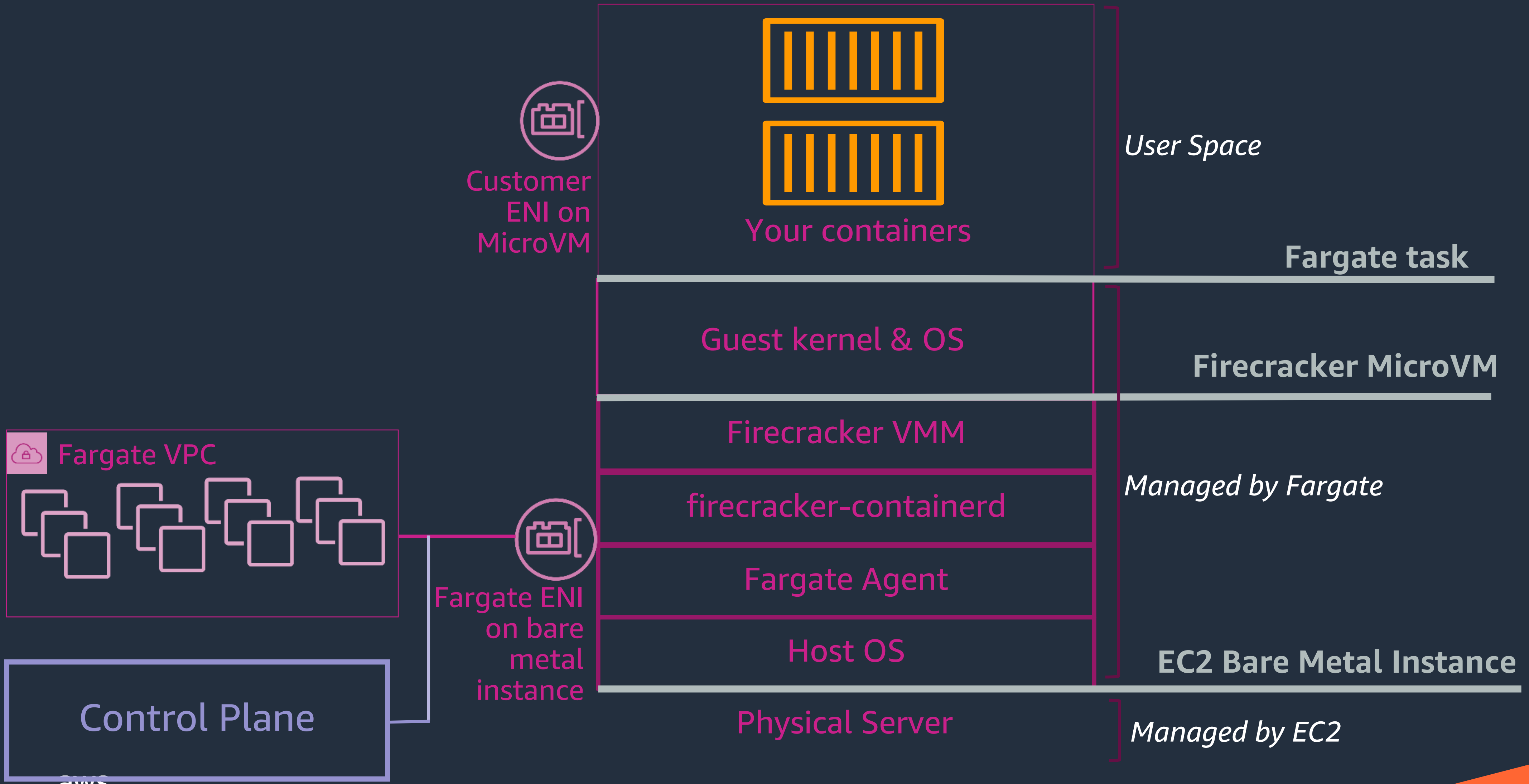


Hardware virtualization allows for safe multi-tenancy

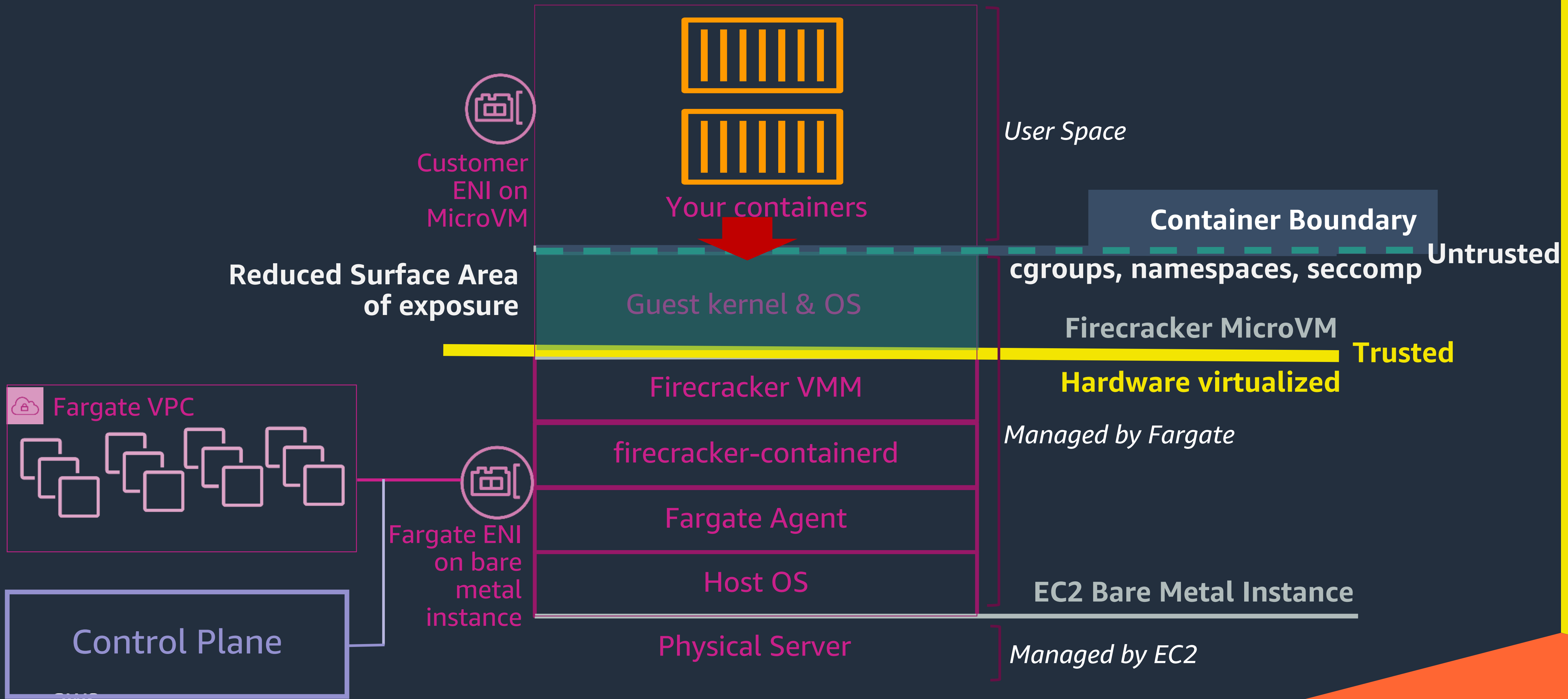
Firecracker is built on KVM, the same hypervisor that EC2 Nitro instances are built on.

Hardware virtualization ensures that tasks from different customers can run safely on the same physical machine.

Fargate on Firecracker networking: A deeper look

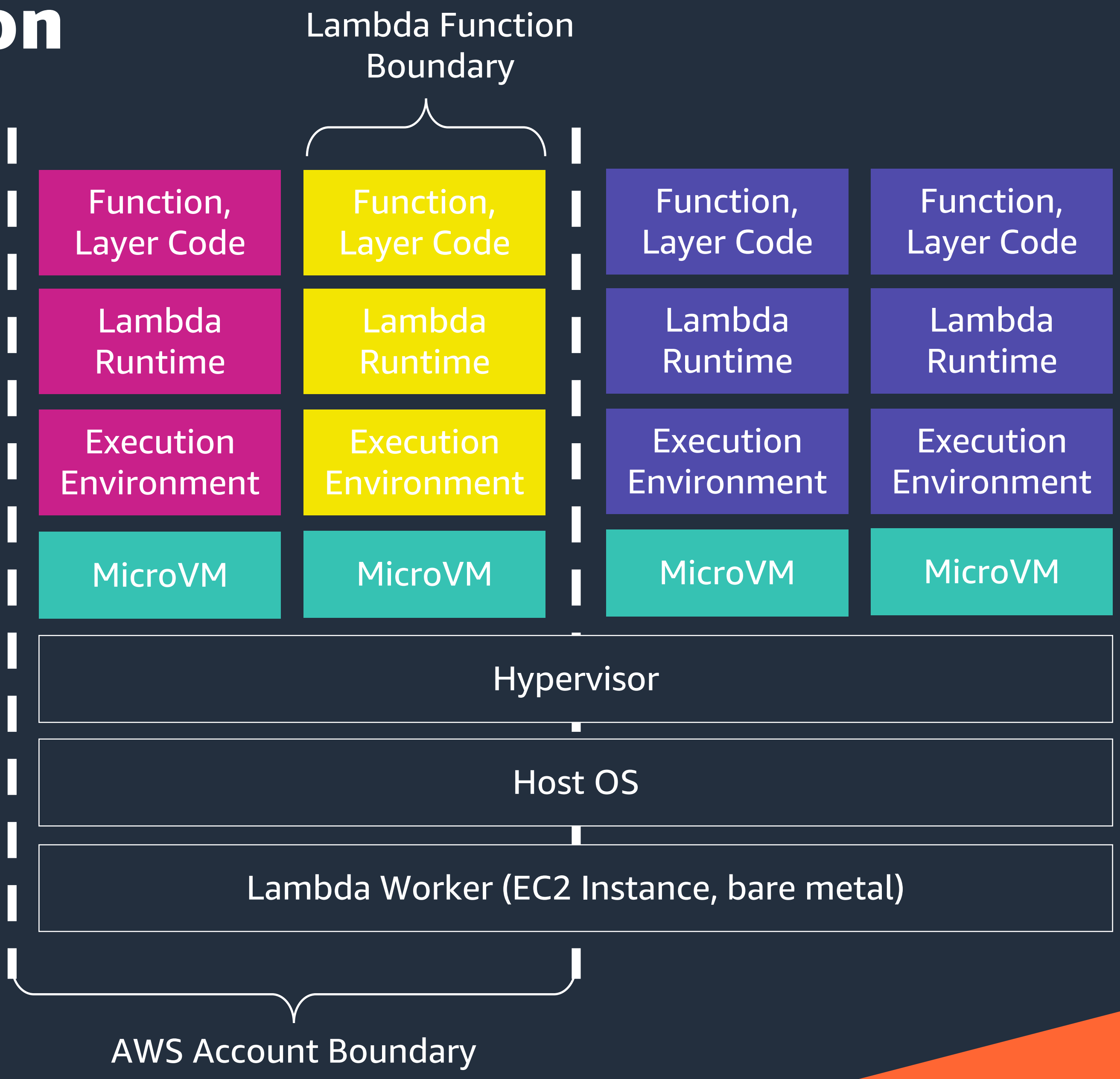


Firecracker enhances isolation of tasks

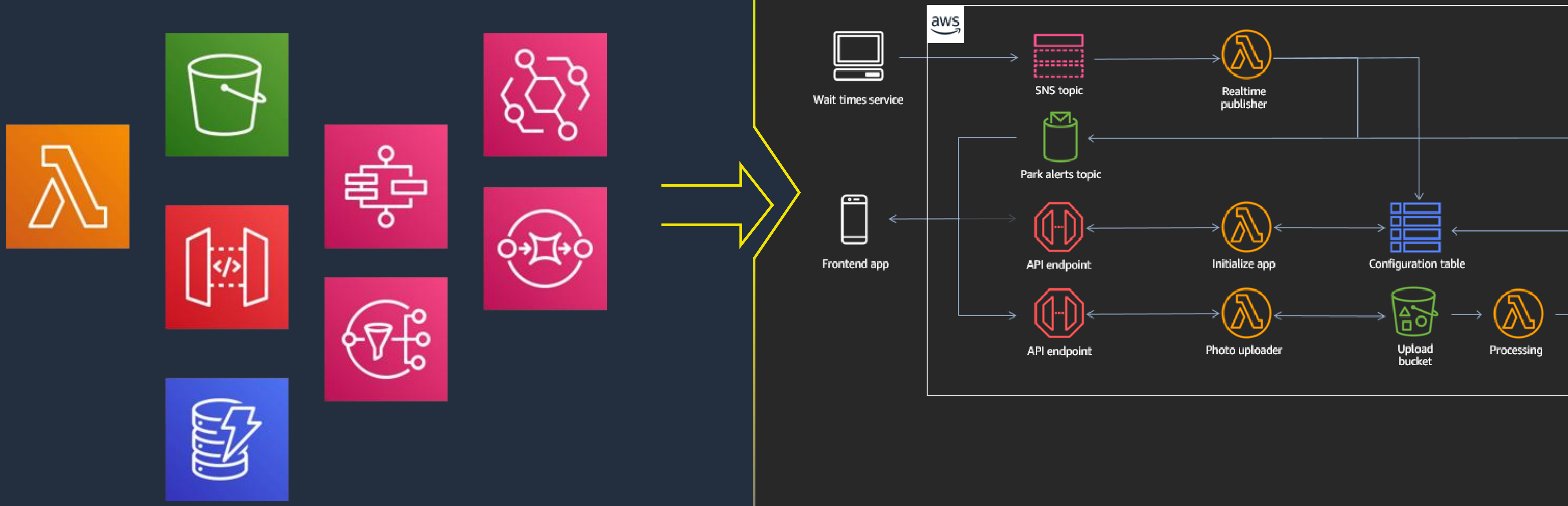


Lambda Function isolation

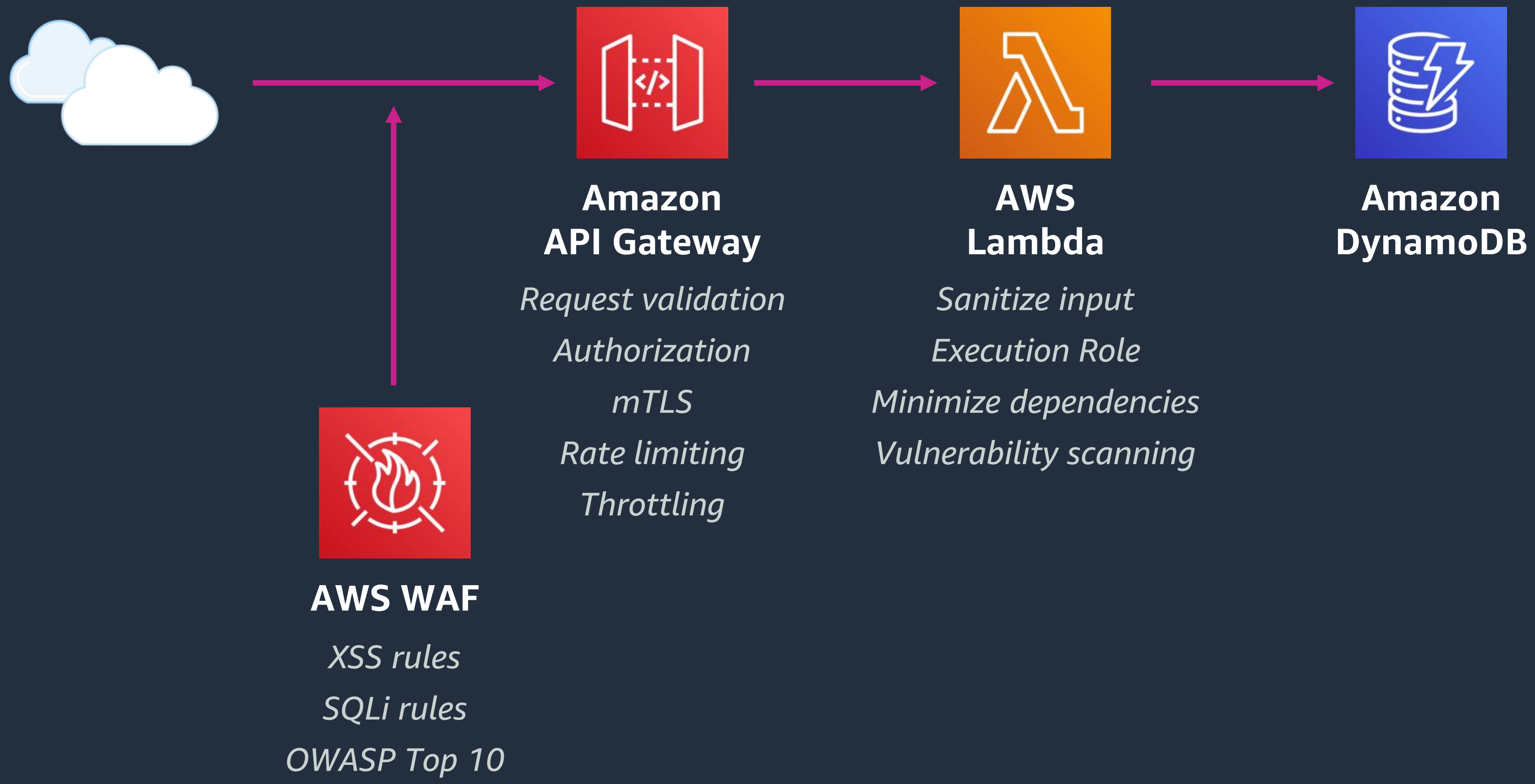
- Each function runs in a dedicated execution environment
  - Each execution environment handles one concurrent invocation
- Execution environment may be reused between invocations
  - Use caution when storing sensitive data in memory or /`tmp`
- AWS maintains runtime and execution environment
  - Patching, etc.
  - Does not apply to container packaging



Serverless architectures are small pieces, loosely joined



Securing a Serverless web service



AWS WAF

XSS rules

SQLi rules

OWASP Top 10

Amazon API Gateway

Request validation

Authorization

mTLS

Rate limiting

Throttling

AWS Lambda

Sanitize input

Execution Role

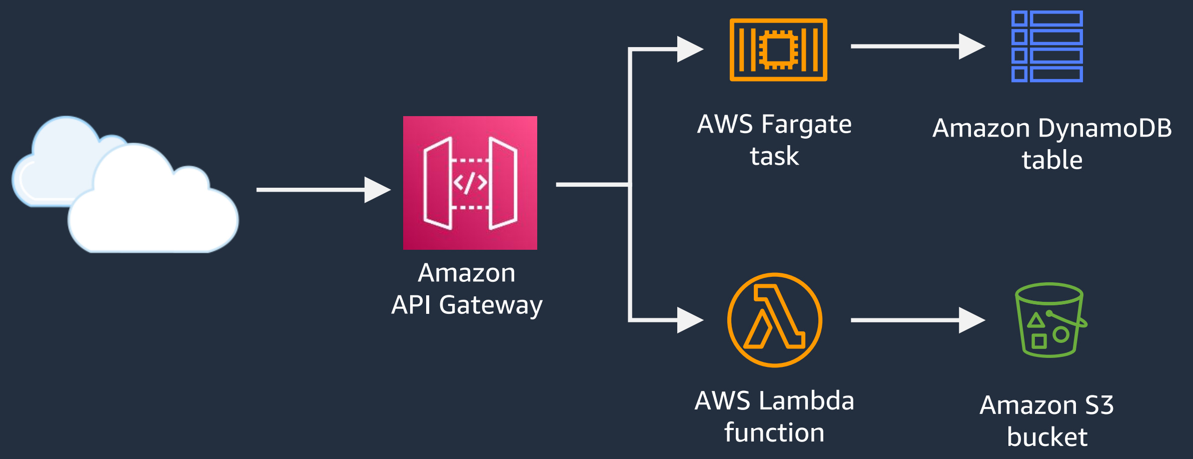
Minimize dependencies

Vulnerability scanning

Amazon DynamoDB

Common ask: How do I secure access to my API? Are API keys good enough?

- Options for authorization:
  - IAM
  - Cognito user pool/JWT
  - Lambda authorizer
- Can be used with:
  - AWS WAF
  - Resource policies
  - Mutual TLS (mTLS)

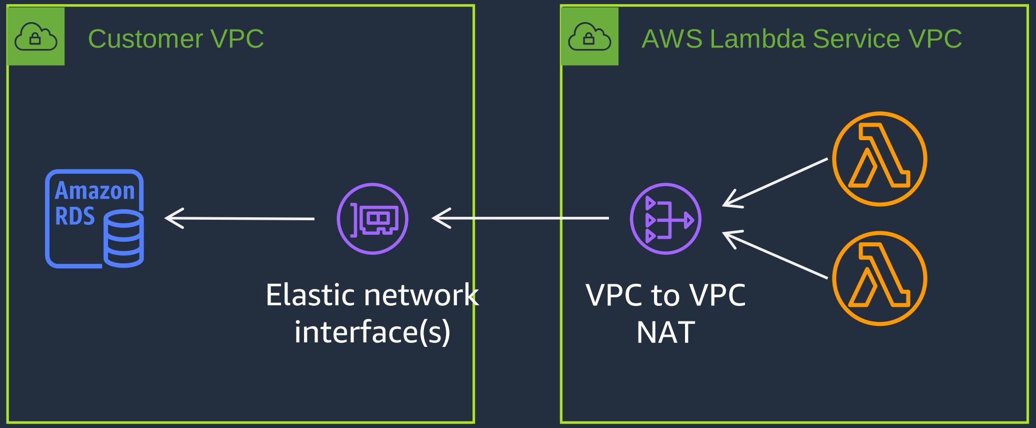


Common ask: Should my Lambda function be VPC-enabled?

- Lambda functions always run in VPCs owned by the Lambda service team
  - When VPC enabled, configured with access to your VPC via an ENI

- Lambda functions are always invoked via `Invoke` action
  - Access controlled by AWS IAM

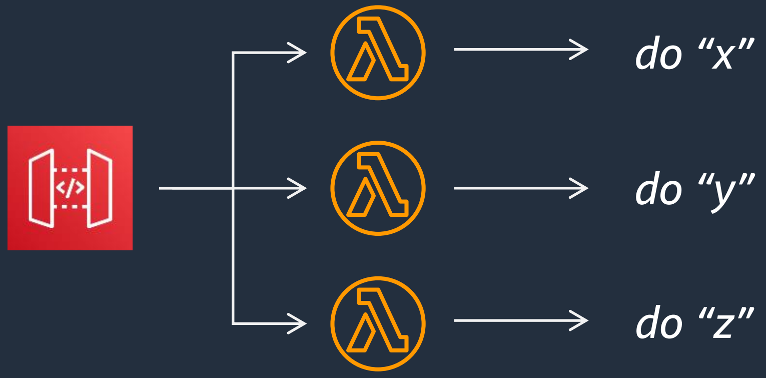
- Answer: Only if your function:
  - Needs access to resources in the VPC
  - Desire to restrict outbound network path



Security Principle #4: Secure Your Software Supply Chain

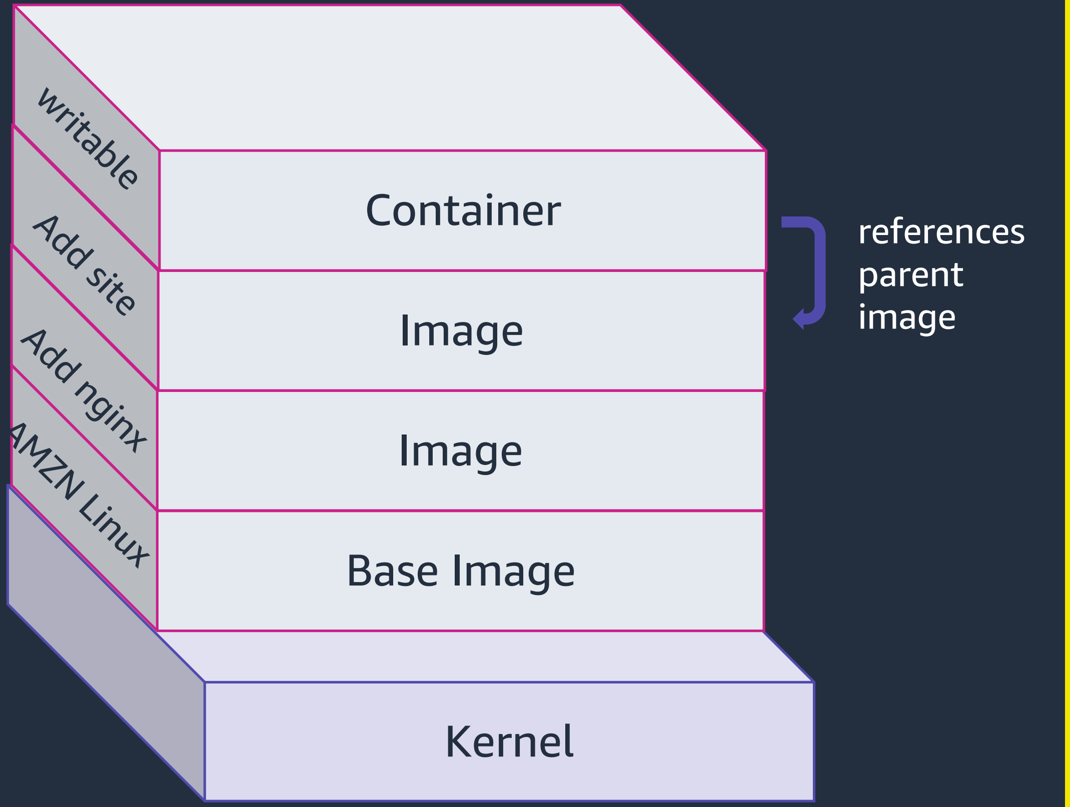
Security principle #4: Secure Your Software Supply Chain

- Keep it simple
  - Prefer single responsibility
  - Easier to debug; cleaner IAM privileges
- Never hardcode secrets in code
  - Use AWS Secrets Manager, Parameter Store
  - Again, never…
- Leverage code and vulnerability scanning
  - Don’t forget dependencies



Components of the software supply chain

- Base image*
- Language runtime*
- Open source, third-party packages
- Your code



* May be supplied and/or managed by AWS

Managing dependencies is key

- Understand you dependencies: https://deps.dev/
- Minimize dependencies
- Keep dependencies up-to-date to reduce risk and effort
- Software Bill of Material (SBOM)
- Leverage dependency check tools, such as:
  - OWASP
  - Protego
  - Snyk
  - Twistlock
  - Puresec

Build secure container images for Fargate and Lambda

Minimizing the attack surface

- Create images from Scratch
- Create minimal images (docker-slim)
- Use distro-less images without package manager or shell
- Run the application as a non-root user
- “Defang” your containers
- Lint your Dockerfiles with Dockle or Hadolint
- Scan your images for vulnerabilities (CVEs)

Securing your code

Educate about writing secure code

Perform static code analysis (whitebox testing)

Perform dynamic security testing

- Proactively inject faults into the application
- Fuzz testing

Thank you!